The AMD DAL Display Driver

PRESENTED BY HARRY WENTLAND

AGENDA

About DAL and the Display Solution Team

DAL3 Design

DAL3 Walkthrough

Challenges & Takeaway

Questions

About DAL and the Display Solution Team

THE TEAM



MAIN LINUX® CONTRIBUTORS

- Harry Wentland



- Andrey Grodzovsky



- Jordan Lazare



- Vitaly Prosyak



- Eric Yang



AMD

A BRIEF HISTORY

DAL (Display Abstraction Layer) Team

- 1998-2008: DAL 1
  - DVI, VGA, LVDS
  - C code architected for Windows 98
- 2008-2015: DAL 2
  - DP1.1, HDMI® 1.x, HDCP 1.x
  - C++ code architected for Windows® Vista. Leveraged into AMD Catalyst™ closed source driver in user mode
  - Later new concepts retrofitted into architecture
    - HW layer composition
    - Dynamic refresh rate (FreeSync)
    - DP1.2 MST

- DAL name is somewhat historic
- We're trying to get away from it to
  - Avoid confusion and
  - Set right direction
- Maybe just "Display Core"

ABOUT DAL

- 2015+: DAL 3
  - Eco-system, HW and OS evolving.
  - New architecture is needed to comprehend the new concepts
    - DRM Atomic mode setting
    - Dynamic refresh rate
    - HW composition
    - Management of shared HW resource
    - Compression
    - Wide Gamut Display and HDR support
  - DP 1.4, HDMI® 2, 4K@120, 5K, 8K, + more…
  - Architecture allow code reuse of HW programming functions
    - Linux®, Android, Windows®
    - Emulation platforms, Diagnostic Test Suite
  - Lots of change to accommodate Linux® Kernel
    - C, less layer, gradually remove abstraction
    - More to come…

Team focuses on providing display end to end solutions on multiple platforms

AMD

DAL2 in AMD Catalyst™

- User space driver
- Closed-source C++
- Windows based architecture retrofitted into AMD Catalyst™ for Linux®

DAL3 in AMD GPU-Pro

- Kernel space driver
- Open source C code
- Native Linux® Kernel support
- Enable early support for new ASIC with high quality

SOME OF WHAT OUR TEAM DOES

- Proactively working with ecosystem and standard bodies
  - VESA, CTA
  - Display inter-operability
  - Compliance requirement
  - Attend plug test and incorporated inter-operability fixes

Inter-op and good user experience is beyond just code

DAL3 Design

DAL3 DESIGN

AMD

Platform policies

OS specific handling
- cursor, surface, composition
- Mode reporting and setting
- Display Capability (EDID, timing)
  - Virtual target (tiled display)
  - Display Adjustment
- Use DRM functionality where available or contribute

Windows® Display Manager

Linux® (amdgpu) Display Manager

HW diagnostic

AMD RADEON GRAPHICS

Display Core

- DCE 11
- DCE 11.2
- DC...

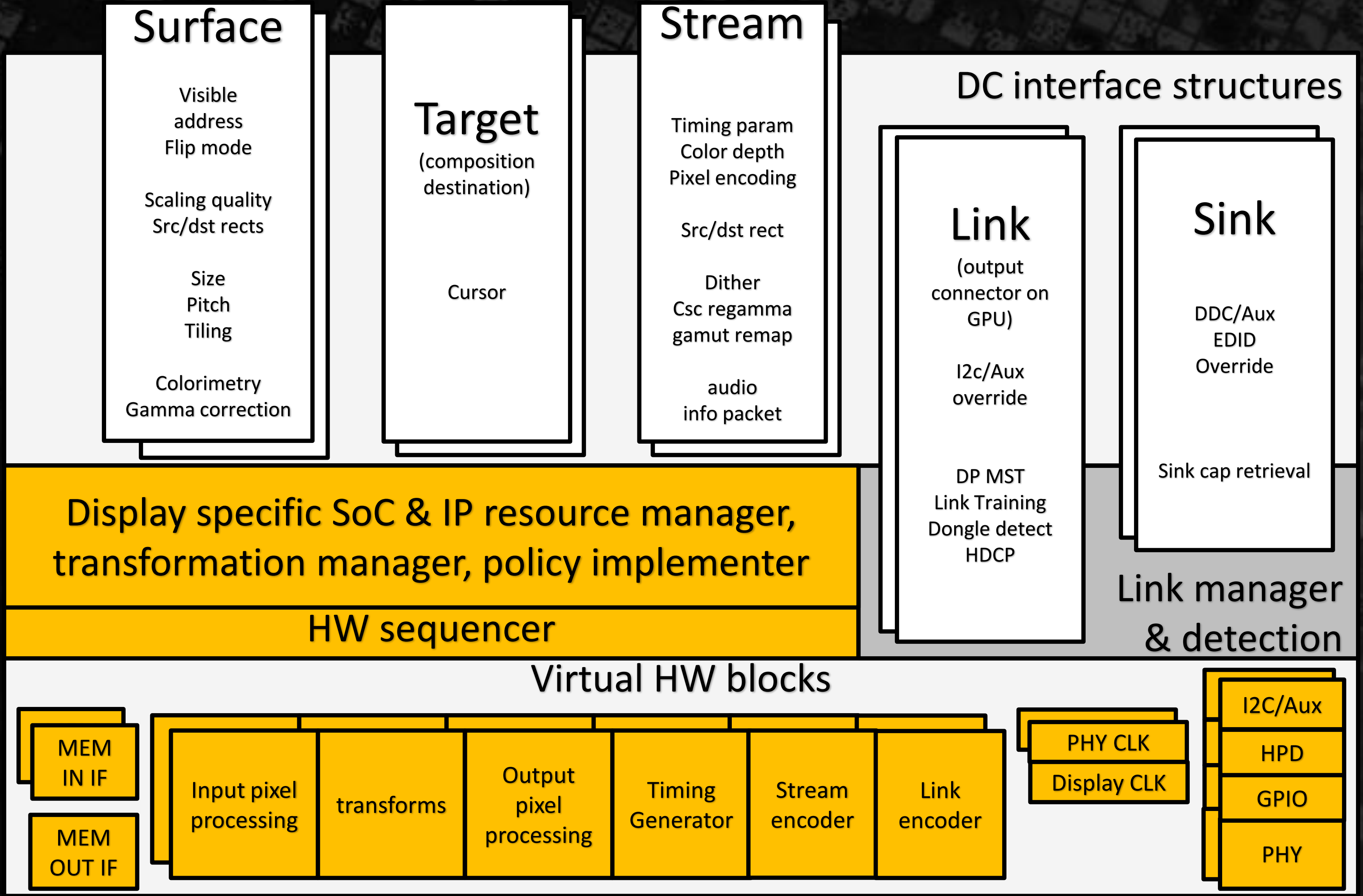
VESA

DisplayPort

HDMI™ HIGH-DEFINITION MULTIMEDIA INTERFACE

AMD FREESYNC TECHNOLOGY

DAL3 DESIGN – DISPLAY CORE



AMD

BENEFITS OF DC/DM DESIGN

- Leverage hundreds of hours of testing coverage
  - Across different display types
    - DP MST, HDMI 2.0, FreeSync
    - 4k, 5k modes. 120Hz, 144Hz
  - Multiple display combinations. Up to 6 displays
  - Daily coverages. Manual + automation

DC = Display Core

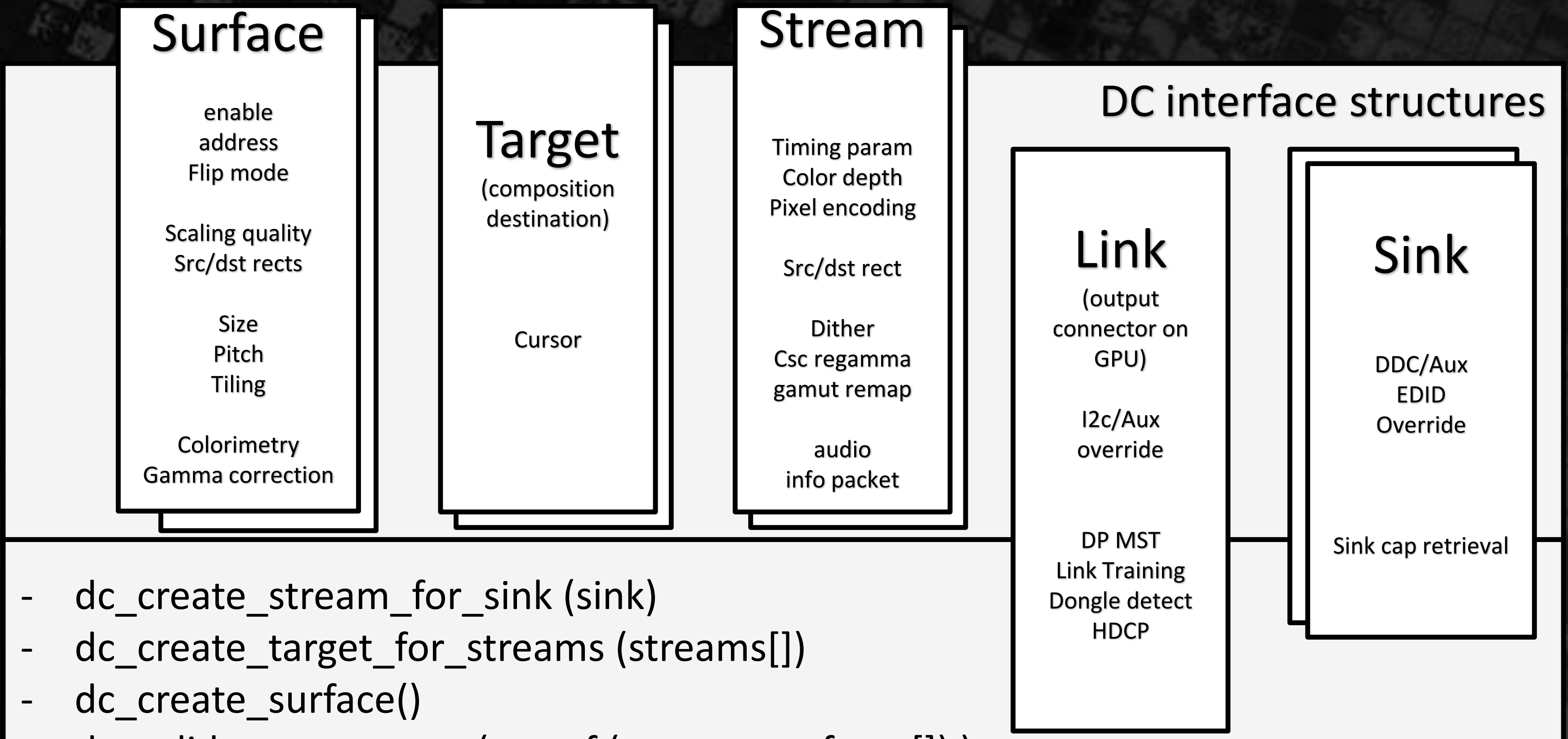
DM = Display Manager

Common code base benefits both Linux® and other OS drivers

DAL3 Walkthrough

DAL3 HW CORE LAYER DESIGN

AMD



- dc_create_stream_for_sink (sink)
- dc_create_target_for_streams (streams[])
- dc_create_surface()
- dc_validate_resources (set of (target + surfaces[]))
- dc_commit_targets (targets[])
- dc_commit_surfaces_to_target(surfaces[], target)

DM_ATOMIC_COMMIT

```
if (!async) {
        ret = drm_atomic_helper_prepare_planes(dev, state);
        if (ret)
                return ret;
}
drm_atomic_helper_swap_state(dev, state);
drm_atomic_helper_update_legacy_modeset_state(dev, state);
```

DM_ATOMIC_COMMIT

```
/* update changed items */
for_each_crtc_in_state(state, crtc, old_crtc_state, i) {
      action = get_dm_commit_action(new_state);
      case DM_COMMIT_ACTION_SET: {
            new_target = create_target_for_sink(
                              aconnector,
                              &crtc->state->mode,
                              dm_state);
            if (!new_target)
                  break;
            if (acrtc->target)
                  remove_target(adev, acrtc);

            new_crtcs[new_crtcs_count] = acrtc;
            new_crtcs_count++;

            acrtc->target = new_target;
            acrtc->enabled = true;
            acrtc->hw_mode = crtc->state->mode;
```

DM_ATOMIC_COMMIT

```
list_for_each_entry(crtc, &dev->mode_config.crtc_list, head) {

        struct amdgpu_crtc *acrtc = to_amdgpu_crtc(crtc);

        if (acrtc->target) {
                commit_targets[commit_targets_count] = acrtc->target;
                ++commit_targets_count;
        }
}

/* DC is optimized not to do anything if 'targets' didn't change. */
dc_commit_targets(dm->dc, commit_targets, commit_targets_count);
```

DC_COMMIT_TARGETS

```
if (false == targets_changed(core_dc, targets, target_count))
        return DC_OK;
result = core_dc->res_pool->funcs->validate_with_context(
                        core_dc, set, target_count, context);
pplib_apply_safe_state(core_dc);
if (!dcb->funcs->is_accelerated_mode(dcb))
        core_dc->hwss.enable_accelerated_mode(core_dc);
if (result == DC_OK) {
        result = core_dc->hwss.apply_ctx_to_hw(core_dc, context);
}
```

DCE110_VALIDATE_WITH_CONTEXT

```
if (!dce110_validate_surface_sets(set, set_count))
        return DC_FAIL_SURFACE_VALIDATE;

context->res_ctx.pool = dc->res_pool;

for (i = 0; i < set_count; i++) {
        context->targets[i] = DC_TARGET_TO_CORE(set[i].target);
        context->target_count++;
}

result = resource_map_pool_resources(dc, context);

if (result == DC_OK)
        result = resource_map_clock_resources(dc, context);
```

DCE110_VALIDATE_WITH_CONTEXT

```
if (!resource_validate_attach_surfaces(
                set, set_count, dc->current_context, context)) {
        DC_ERROR("Failed to attach surface to target!\n");
        return DC_FAIL_ATTACH_SURFACES;
}

if (result == DC_OK)
        result = validate_mapped_resource(dc, context);

if (result == DC_OK)
        result = resource_build_scaling_params_for_context(dc, context);

if (result == DC_OK)
        result = dce110_validate_bandwidth(dc, context);

return result;
```

DC_COMMIT_TARGETS

```
if (false == targets_changed(core_dc, targets, target_count))
        return DC_OK;
result = core_dc->res_pool->funcs->validate_with_context(
                        core_dc, set, target_count, context);
pplib_apply_safe_state(core_dc);
if (!dcb->funcs->is_accelerated_mode(dcb))
        core_dc->hwss.enable_accelerated_mode(core_dc);
if (result == DC_OK) {
        result = core_dc->hwss.apply_ctx_to_hw(core_dc, context);
}
```

APPLY_CTX_TO_HW

```
	for (i = 0; i < MAX_PIPES; i++)
		if (!pipe_ctx->stream ||
				pipe_need_reprogram(pipe_ctx_old, pipe_ctx))
			reset_single_pipe_hw_ctx(
				dc, pipe_ctx_old, dc->current_context);

	for (i = 0; i < MAX_PIPES; i++) {

		if (pipe_ctx->stream == pipe_ctx_old->stream) {
			if (pipe_ctx_old->clock_source != pipe_ctx->clock_source)
				dc->hwss.crtc_switch_to_clk_src(
						pipe_ctx->clock_source, i);
			continue;
		}
		dc->hwss.enable_display_power_gating(
				dc, i, dcb,
				PIPE_GATING_CONTROL_DISABLE);
	}
```

APPLY_CTX_TO_HW

```
set_safe_displaymarks(&context->res_ctx);

if (context->bw_results.dispclk_khz
        > dc->current_context->bw_results.dispclk_khz)
        set_display_clock(context);

for (i = 0; i < MAX_PIPES; i++) {
        if (pipe_ctx->stream == NULL)
                continue;
        if (pipe_ctx->stream == pipe_ctx_old->stream)
                continue;
        status = apply_single_controller_ctx_to_hw(
                        pipe_ctx,
                        context,
                        dc);
}
```

APPLY_SINGLE_CONTROLLER_CTX_TO_HW

```
	if (!pipe_ctx_old->stream) {
		pipe_ctx->tg->funcs->set_blank(pipe_ctx->tg, true);

		pipe_ctx->clock_source->funcs->program_pix_clk(
				pipe_ctx->clock_source,
				&pipe_ctx->pix_clk_params,
				&pipe_ctx->pll_settings))

		pipe_ctx->tg->funcs->program_timing(
				pipe_ctx->tg,
				&stream->public.timing,
				true);
	}

	pipe_ctx->mi->funcs->allocate_mem_input(
					pipe_ctx->mi,
					stream->public.timing.h_total,
					stream->public.timing.v_total,
					stream->public.timing.pix_clk_khz,
					context->target_count);
```

TG->FUNCS->SET_BLANK

```
set_reg_field_value(
        value,
        1,
        CRTC_BLANK_CONTROL,
        CRTC_BLANK_DATA_EN);
dm_write_reg(tg->ctx, CRTC_REG(mmCRTC_BLANK_CONTROL), value);

for (counter = 0; counter < 100; counter++) {
        value = dm_read_reg(tg->ctx, CRTC_REG(mmCRTC_BLANK_CONTROL));

        if (get_reg_field_value(
                value,
                CRTC_BLANK_CONTROL,
                CRTC_BLANK_DATA_EN) == 1 &&
                get_reg_field_value(
                value,
                CRTC_BLANK_CONTROL,
                CRTC_CURRENT_BLANK_STATE) == 1)
                break;

        msleep(1);
}
```

APPLY_SINGLE_CONTROLLER_CTX_TO_HW

```
if (!pipe_ctx_old->stream)
        pipe_ctx->tg->funcs->enable_crtc(
                pipe_ctx->tg))

bios_parser_crtc_source_select(pipe_ctx))

pipe_ctx->opp->funcs->opp_set_dyn_expansion(
                pipe_ctx->opp,
                COLOR_SPACE_YCBCR601,
                stream->public.timing.display_color_depth,
                pipe_ctx->stream->signal);

pipe_ctx->opp->funcs->opp_program_fmt(
                pipe_ctx->opp,
                &stream->bit_depth_params,
                &stream->clamping);
```

APPLY_SINGLE_CONTROLLER_CTX_TO_HW

```
stream->sink->link->link_enc->funcs->setup(
                stream->sink->link->link_enc,
                pipe_ctx->stream->signal);

if (dc_is_dp_signal(pipe_ctx->stream->signal))
        pipe_ctx->stream_enc->funcs->dp_set_stream_attribute(
                pipe_ctx->stream_enc,
                &stream->public.timing);
/* same for hdmi and dvi */

color_space_to_black_color(
                stream->public.output_color_space, &black_color);
pipe_ctx->tg->funcs->set_blank_color(
                pipe_ctx->tg,
                &black_color);
```

APPLY_SINGLE_CONTROLLER_CTX_TO_HW

```
	if (!pipe_ctx_old->stream) {
		core_link_enable_stream(pipe_ctx);

		if (dc_is_dp_signal(pipe_ctx->stream->signal))
			unblank_stream(pipe_ctx,
				&stream->sink->link->public.cur_link_settings);
	}

	if (!pipe_ctx_old || memcmp(&pipe_ctx_old->scl_data,
				&pipe_ctx->scl_data,
				sizeof(struct scaler_data)) != 0)
		program_scaler(dc, pipe_ctx);

	return DC_OK;
```

APPLY_CTX_TO_HW

```
for (i = 0; i < MAX_PIPES; i++) {
        if (context->res_ctx.pipe_ctx[i].audio != NULL) {

                build_audio_output(pipe_ctx, &audio_output);
                dal_audio_setup(
                                pipe_ctx->audio,
                                &audio_output,
                                &pipe_ctx->stream->public.audio_info))
                if (!programmed_audio_dto) {
                        dal_audio_setup_audio_wall_dto(
                                pipe_ctx->audio,
                                pipe_ctx->stream->signal,
                                &audio_output.crtc_info,
                                &audio_output.pll_info);
                        programmed_audio_dto = true;
                }
        }
}
```

APPLY_CTX_TO_HW

```
dc->hwss.set_displaymarks(dc, context);

switch_dp_clock_sources(dc, &context->res_ctx);

return DC_OK;
```

DC_COMMIT_TARGETS

```
program_timing_sync(core_dc, context);
for (i = 0; i < context->target_count; i++) {
      if (context->target_status[i].surface_count > 0)
            target_enable_memory_requests(dc_target,
                          &core_dc->current_context->res_ctx);
}
pplib_apply_display_requirements(core_dc,
              context, &context->pp_display_cfg);
```

DM_ATOMIC_COMMIT

AMD

```
/* update planes when needed */
for_each_plane_in_state(state, plane, old_plane_state, i) {
        if (!fb || !crtc || !crtc->state->planes_changed ||
                !crtc->state->active)
                continue;
        action = get_dm_commit_action(crtc->state);

        if (!page_flip_needed(plane_state, old_plane_state, true) ||
                        action == DM_COMMIT_ACTION_DPMS_ON ||
                        action == DM_COMMIT_ACTION_SET) {
                wait_while_pflip_status(adev, acrtc, pflip_pending_predicate);

                dm_dc_surface_commit(dm->dc, crtc);
        }
}

for (i = 0; i < new_crtcs_count; i++) {
        manage_dm_interrupts(adev, acrtc, true);
        dm_crtc_cursor_reset(&acrtc->base);

}
```

DM_ATOMIC_COMMIT

```
/* Page flip if needed */
for_each_plane_in_state(state, plane, old_plane_state, i) {
        struct drm_plane_state *plane_state = plane->state;
        struct drm_crtc *crtc = plane_state->crtc;
        struct amdgpu_crtc *acrtc = to_amdgpu_crtc(crtc);
        struct drm_framebuffer *fb = plane_state->fb;

        if (page_flip_needed(plane_state, old_plane_state, false)) {
                ret = amdgpu_crtc_page_flip(crtc,
                                            fb,
                                            crtc->state->event,
                                            acrtc->flip_flags);
        }
}
```

Challenges & Takeaway

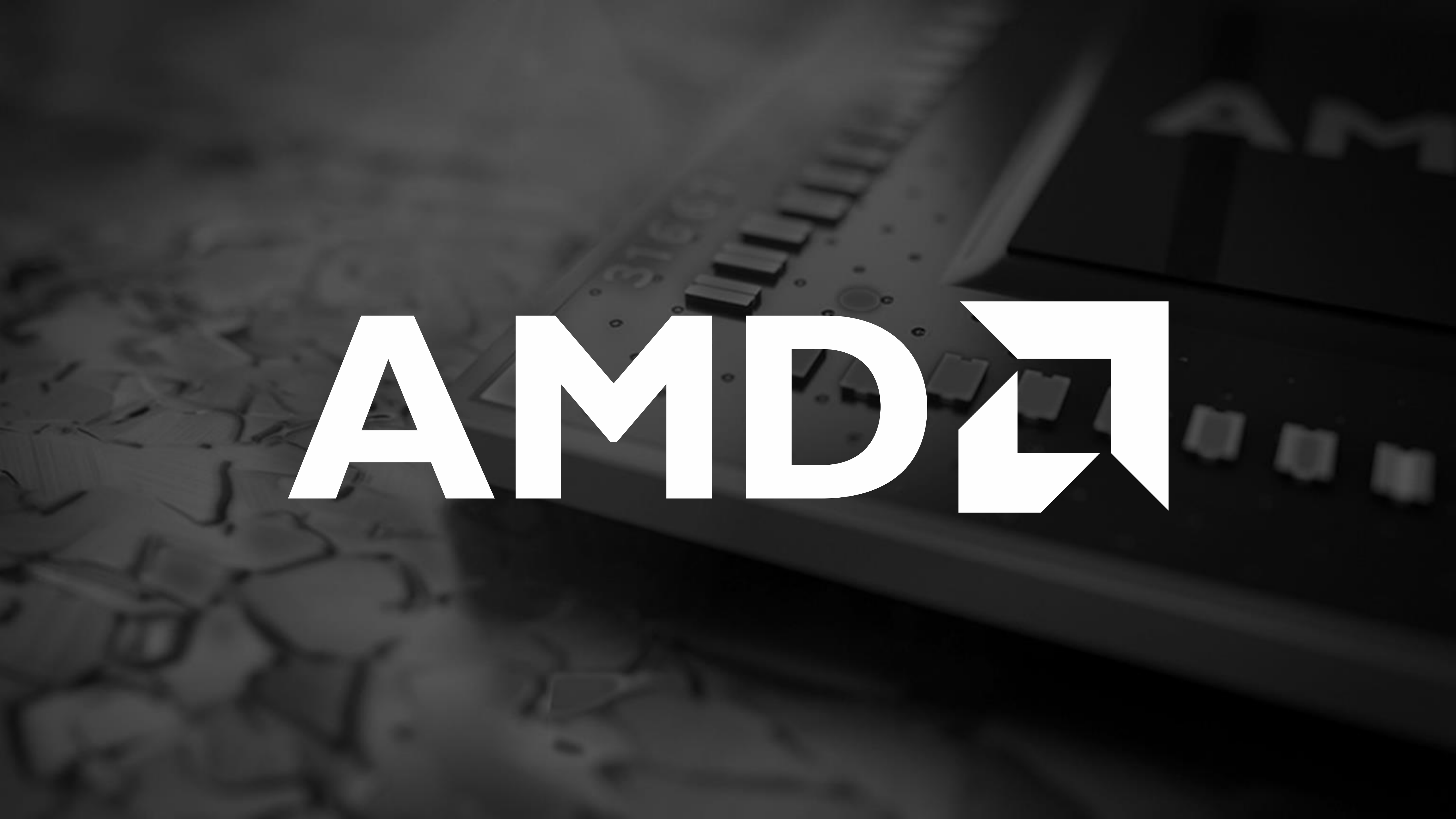
CHALLENGES

- Delivering new feature / functionality with DRM dependency in Distro
  - e.g. Ubuntu uses kernel 4.4
  - Requires us to work on feature a year or half ahead
    - What if spec isn't even clear at that point?
  - Some example
    - 5K DisplayID parsing code is in Kernel 4.7
    - HDMI® 2.0 CEA extension parsing for 4K@60 modes
    - We started patching MST early this year but we still don’t have MST working in Ubuntu

TAKEAWAY

- DAL3 Architecture
  - Designed with Linux® in mind
  - Ready for future technologies we anticipate from the industry
- Committed to Linux®
- Willing to work together
- Eager to share our experience in the industry with the community

Questions


AMD

DISCLAIMER & ATTRIBUTION

ATTRIBUTION